10. Übung zur Vorlesung "Datenbanken" im Sommersemester 2007
– mit Musterlösungen –

Prof. Dr. Gerd Stumme, Dr. Andreas Hotho, Dipl.-Inform. Christoph Schmitz

9. Juli 2007

Aufgabe 1 – Transaktionsverarbeitung

Betrachten Sie als Beispiel ein Kino. Dort werden an mehreren Kassen Plätze reserviert, Karten für bestimmte Plätze verkauft (gegen Bargeld oder online per Kreditkarte mit Karte zum Selberausdrucken), freie Sitzplätze verwaltet, nicht benötigte Karten zurückgenommen, Cola und Popcorn gezapft, usw.

Nennen Sie je zwei Beispiele, wofür hier die Eigenschaften A, C, I, D von Transaktionen nützlich sind.

- Eigenschaft A (Atomicity)
  - die Vorgänge "Erstellen einer Karte", "Abbuchung des Kartenpreises" und "Reservieren eines Sitzplatzes" erfolgen entweder alle, oder es erfolgt keiner der drei
  - ein Kunde, der mehrere Plätze reservieren will, bekommt entweder alle oder keinen davon
- Eigenschaft C (Consistency)
  - die Summe der verkauften und der nicht verkauften Plätze ist aus Sicht jeder Transaktion gleich der Gesamtzahl der Plätze
  - es verschwindet kein Geld oder wird neu erschaffen: nach jeder Transaktion ist die Summe des Geldes auf den Konten der Kunden und des Kinos konstant
- Eigenschaft I (Isolation)
  - jeder Sitzplatz wird nur höchstens einmal verkauft, auch wenn mehrere Kassierer gleichzeitig Reihe 8, Platz 15 verkaufen wollen
  - Geldzugänge auf das Konto des Kinos der Art:
    1. BOT

2. r(Kontostand, k)
3. k = k + Preis einer Karte
4. w(Kontostand, k)
5. COMMIT

überschreiben sich nicht gegenseitig die Kontostände, so daß das Geld auch tatsächlich in vollem Umfang ankommt

- Eigenschaft D (Durability)
  - ein reservierter oder verkaufter Sitzplatz behält diesen Status auch bei (es sei denn, die Karte wird in einer weiteren Transaktion zurückgenommen)
  - umgebuchtes Geld verschwindet nicht wieder vom Konto des Kinos

Aufgabe 2 – Logging und Recovery

Obwohl Abstürze natürlich auch während des Wiederanlaufs geschehen können, enthalten CLRs keine Undo-Information. Warum nicht?

- Das Feld *UndoNextLSN* zeigt an, wieweit die Recovery fortgeschritten ist.
- Daher brauchen die von CLRs dokumentierten Operationen nicht rückgängig gemacht zu werden: bei einem Crash während der Recovery ist durch die *UndoNextLSN*-Felder der CLRs klar, welche Operationen schon zurückgesetzt wurden.
- *Bemerkung:* Es wird also nur höchstens ein CLR pro Logeintrag benötigt, selbst bei mehrfachen Versuchen zum Wiederanlauf.

Aufgabe 3 – Einbringstrategien

- Sie möchten ein DBMS ohne Undo- und Redo-Log implementieren. Welche Einbringstrategien (force/¬force, steal/¬steal) müßten Sie dazu wählen? (Begründung!)

  Man müßte ¬steal, force wählen. Ohne Redo-Log kann man kein ¬force zulassen, da dann evtl. noch nicht herausgeschriebene Daten verloren gehen würden. Ohne Undo-Log kann man nicht steal wählen, da sonst "schmutzige" Änderungen von nicht erfolgreich beendeten Transaktionen ohne die Möglichkeit einer Rücknahme herausgeschrieben werden würden.

- Wie können beim Twin-Block-Verfahren (genauso beim Schattenspeicherkonzept) atomare Schreiboperationen auf eine Seite – d. h. es befindet sich zu jeder Zeit entweder der alte oder der neue Stand der Seite auf dem Hintergrundspeicher – sichergestellt werden?

Zu jeder Seite speichert man sich in einem Bit, welche Kopie gerade gültig ist. Ein Bit auf die Platte zu schreiben ist atomar möglich: zu jedem Zeitpunkt ist es entweder 0, oder es ist 1.

Man schreibt also erst die Daten in die ungenutzte Kopie und ändert dann dieses Bit.

- Wie könnte man atomare Schreiboperationen auf *mehrere* Seiten realisieren?

  Man arbeitet mit zwei Kopien des Bitvektors, die die gerade aktuellen Kopien der Seiten anzeigt. Wie bei der vorigen Aufgabe schaltet man mit einem Bit um, diesmal allerdings zwischen den Kopien dieses Bitvektors und damit zwischen den Kopien der Seiten.

Aufgabe 4 - Transaktionsverarbeitung

1. Nehmen Sie an, der Puffer habe nur eine Kapazität von einer Seite.

   Zeigen Sie, was die folgende Transaktion im Puffer, auf dem Hintergrundspeicher und im Log hinterlässt für die Strategie *force, steal*.

T1	T2	Puffer	Log [LSN,T#,P#,Undo,PrevLSN]	Platte
BOT			**[1,T1,BOT,0]**	A, B
r(A,a)		**A**		**A, B**
a' = a + 10		**A'**		**A, B**
	BOT	**A'**	**[2,T2,BOT,0]**	**A, B**
w(A,a')		**A'**	**[3,T1,A,a-=10,1]**	**A, B**
	r(B,b)	**B (steal)**		**A', B**
	b' = b - 10	**B'**		**A', B**
	w(B,b')	**B'**	**[4,T2,B,b+=10,2]**	**A', B**
ABORT			**[5,T1,ABORT,3]**	**A, B**
	COMMIT		**[6,T2,COMMIT,4]**	**A, B'**

   Welche Logeinträge wurden beim Abort von T1 benötigt?

   Um den ursprünglichen Wert der Seite A wiederherzustellen, der beim steal überschrieben wurde, wird der Undo-Log-Eintrag mit LSN 3 benötigt.

2. Die Strategie sei nun *noforce, nosteal*. Nehmen Sie an, daß die Seiten vor dem Crash nicht ausgeschrieben worden sind.

T1	T2	Puffer	Log [LSN, T#, P#, Redo, PrevLSN]	Platte
BOT			**[1,T1,BOT]**	A, B
r(A,a)		**A**		**A, B**
a' = a + 10		**A**		**A, B**
w(A,a')		**A'**	**[2,T1,a+=10,1]**	**A, B**
COMMIT		**A'**	**[3,T1,COMMIT]**	**A, B**
	BOT	**A'**	**[4,T2,BOT]**	**A, B**
	r(A,a)	**A'**		**A, B**
	a' = a - 50	**A'**		**A, B**
	w(A,a')	**A"**	**[5,T2,a-=50,4]**	**A, B**
	ABORT	**A"**	**[6,T2,ABORT]**	**A, B**

Das System stürzt an dieser Stelle ab - wie kann der korrekte Zustand wiederhergestellt werden?

Das Log wird rückwärts durchsucht; Resultat A' der Transaktion 1 kann aus altem Zustand und Log-Eintrag 2 wiederhergestellt werden.

SQL-Übungsaufgabe

1. Kopieren Sie die Tabelle `uni.professoren` samt Daten unter einem anderen Namen in Ihr eigenes Schema! (Tip: `create table ...like ...`; `insert into ...(select ...)`)
2. Fügen Sie eine Spalte für das Gehalt hinzu!
3. Erfinden Sie Gehälter für die Professoren und fügen Sie diese ein.
4. Finden Sie die Professoren, die Gehälter haben, die in einem Bereich von zehn Prozent um das mittlere Professorengehalt liegen.

```
create table myprof like uni.professoren

insert into myprof (select * from uni.professoren)

alter table myprof add column gehalt integer

update myprof set gehalt = 55000 where name = 'Sokrates'
-- usw...

select *
from myprof
where gehalt
between (select 0.9 * avg(gehalt) from myprof)
```

```
and (select 1.1 * avg(gehalt) from myprof)
```

Lösung zur SQL-Aufgabe vom 25.6.

1. Berechnen Sie die Gesamtzahl der Semesterwochenstunden, die die einzelnen Professoren erbringen. Dabei sollen auch die Professoren berücksichtigt werden, die keine Vorlesungen halten.

```
select persnr, name, sum(sws) as swssumme
from professoren left outer join vorlesungen
     on persnr=gelesenvon
group by persnr, name
```

2. Finden Sie die Namen der Studenten, die in keiner Prüfung eine bessere Note als 3.0 hatten.

   a) Berücksichtigen Sie dabei alle Studenten, auch die ohne Prüfung.

```
select name
from studenten
where matrnr not in (
  select matrnr
  from pruefen
  where note < 3.0
)
```

   b) Berücksichtigen Sie nur die Studenten, die schon eine Prüfung abgelegt haben.

```
select name
from studenten s join pruefen p on s.matrnr = p.matrnr
where s.matrnr not in (
  select matrnr
  from uni.pruefen
  where note < 3.0
)
```